dSUMO: Towards a Distributed SUMO

Quentin Bragard[1], Anthony Ventresque[1] and Liam Murphy[1]

[1] Lero@UCD, School of Computer Science and Informatics
University College Dublin, Ireland
Quentin.Bragard@ucdconnect.ie, {Anthony.Ventresque, Liam.Murphy}@ucd.ie

Abstract. Microscopic urban mobility simulations consist of modelling a city's road network and infrastructure, and to run autonomous individual vehicles to understand accurately what is going on in the city. However, when the scale of the problem space is large or when the processing time is critical, performing such simulations might be problematic as they are very computationally expensive applications. In this paper, we propose to leverage the power of many computing resources to perform quicker or larger microscopic simulations, keeping the same accuracy as the classical simulation running on a single computing unit. We have implemented a distributed version of SUMO, called *dSUMO*. We show in this paper that the accuracy of the simulation in SUMO is not impacted by the distribution and we give some preliminary results regarding the performance of dSUMO compared to SUMO.

Keywords: Urban Traffic Simulation, Distributed Simulation, Road Network Partitioning.

1 Introduction

Traffic congestion is a major problem for most urbanised societies leading to delays experienced by commuters, accidents in dense traffic, etc. which costs the society and individuals. If the 86% of the population of the developed countries that are predicted to live in cities by 2050 [8] use their vehicles as we do now, it will only get worse. In Dublin for instance, 34.6% of the active population commute by car [1], even after a huge effort from the municipality to reduce this number (50% in 2000 [14]). Dublin City Council implemented several new policies in the period to obtain this result: incentives to use alternative transport modes, modification of the infrastructure (e.g., cycle lanes, tramway), etc.

In this regard, urban traffic simulations can be useful at design time (road planning) and at run time: (i) the example of Dublin where several policies were implemented in the early 2000s, and led to almost 30% reduction in the number of commuters show that it is possible to have an impact on the city traffic by an adequate urban planning; (ii) correct decisions taken during incidents by emergency personnel can prevent congestion or ease their evacuation. There exist several types of traffic simulations: statistical models which predict certain values (e.g., number of vehicles, travel time) in the road network using stochastic methods; macro-simulation which

consists in modelling the traffic flows (including lane changing for instance); and micro-simulation, representing every vehicle in the model by an agent that makes decisions based on predefined behaviour, context, etc. In this paper we focus on micro-simulations (also known as microscopic simulations) as they have two clear advantages. First, they operate at the vehicle level and thus they allow the user to observe the behaviour of individuals; decision makers can literally see the impact of the infrastructure and their policy on the traffic. (Most solutions now have some sort of GUI which shows with many details the real environment. It is thus very useful for decision makers to understand what is going on, what is not optimal in the system, and what the consequences of their decisions are.) Secondly, they are the most accurate simulation tools, as they model in great detail; the drivers' behaviours, their environment, etc. These two characteristics make micro-simulations the best tool in our scenarios where we target an accurate understanding of the evolution of the traffic with regard to some specific infrastructure design, and some exact short term prediction.

However, both cases are complex: while in the first case the challenge resides in the complexity of the simulated model, the processing time is critical in the second case. Microscopic simulations are not known to perform well for large scale simulations or for fast prediction, which seems to exclude micro-simulations for our scenarios. But, there exists a classical solution to this problem of scale/speed: distributing the processing [19]. So the question is now: can we come up with a distributed version of a traffic simulator which allows to scale up or speed up the simulations?

In this paper, we present a distributed version of the well-known [10] microscopic mobility simulator SUMO [3], that we called *dSUMO*. In dSUMO simulations that normally run on a single CPU of a single machine can run on several cores or a single machine and on remote machines alike. dSUMO clearly targets large scale and very quick simulations that might not be able to run on single machines.

In dSUMO every machine can run various dSUMO nodes (e.g., one per core or one per machine) and nodes distributed on remote machines or local cores alike communicate using sockets. The SUMO instances located in every dSUMO nodes are driven by some dSUMO components (called *Runners* or *Handlers*) and the distribution is for the end-user totally transparent: vehicles are transferred from one dSUMO node to another, simulation can be paused, simulation models are partitioned and each partition allocated to a node, etc. In the rest of this paper we first give an overview of dSUMO (Section 2). Then we present the partitioning methods that we can use for an implementation of dSUMO (Section 3). This is followed by a description of dSUMO architecture, communication protocol and partition borders management (Sections 4, 5 and 6). After that we demonstrate that dSUMO scales-up and speeds-up pretty well, while it does not seem to introduce errors (Section 7). Finally we conclude the paper and give some possible future directions and plans for dSUMO in Section 8.

2 dSUMO: Overview

The goal of a distributed simulator is to run a a single machine (e.g., single CPU) simulator on various machines without anyone really noticing it. For instance, we want here to run several instances of SUMO on different CPUs of a single machine or remote machines, and we want to make sure this is transparent for the users: for them there is no apparent difference between the two modes of running SUMO. Every bit that makes the system distributed, the localisation of the running instances, the distribution of data, etc., needs to be hidden from the user. To achieve this objective, dSUMO has to address several problems:

- partitioning of the data: i.e., how do we split a single environment (map, cars, etc.) to fit in the running nodes we have?
- communication between instances of the running system: i.e., what protocols do we need when dSUMO nodes want to send meaningful information to their neighbours?
- synchronisation between dSUMO nodes: i.e., what information is needed to be exchanged between partitions while they run their individual simulations and their neighbours need some of their information (position of cars when they are close to the borders)?

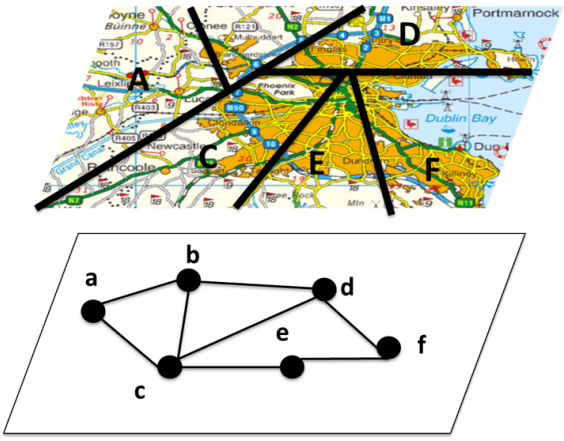


Fig. 1. A map is partitioned and each region is assigned to a computing node in dSUMO.

Figure 1 describes the partitioning of a map and the allocation of each partition to a computing node in dSUMO: partition **A** in the map shown in the upper half of the figure is managed by node **a** in the bottom part of the same figure, etc. Actually we are not interested in the map itself, but in the road network, and if a vehicle travels on a road segment that crosses the border between two partitions, e.g., **A** to **B**, then in dSUMO a message is sent from the origin node (**a**) to the destination node (**b**). These messages describe the vehicle characteristics (type, speed, position, destination, route, etc.) and generate a new vehicle in the destination node, **b** in our example of a vehicle

crossing the border **A/B**. Section 3 describes the state-of-the-art partitioning algorithms that can be used for the same task in dSUMO, while Section 4 shows the technical composition of each dSUMO node.

The synchronisation protocol of dSUMO is depicted in Figure 2. SUMO is a step based simulation and we do not modify this behaviour. At the end of each step every node sends synchronisation messages to all its neighbours: vehicles that are crossing the borders, position of vehicles that have just crossed as the car following model needs it, etc. Then, when a node receives a message from all its neighbours, it knows they have all finished their step, and it can move on to the next step. This gives to the system some interesting properties (reliability, failure detection, etc.) that we describe in Section 5.

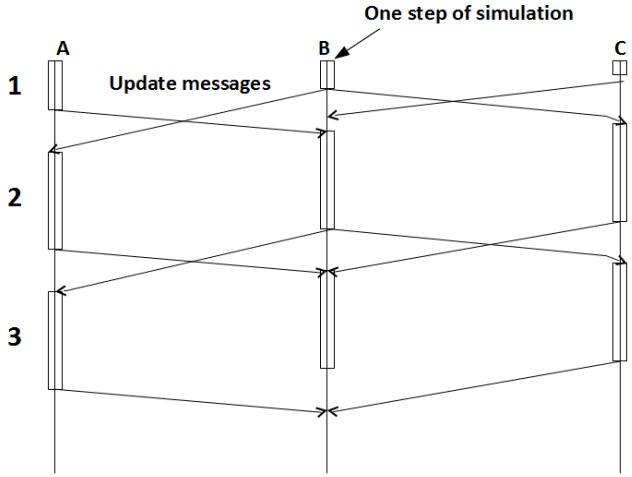


Fig. 2. Synchronisation protocol of dSUMO.

Vehicles in SUMO adapt their behaviour according to the vehicles preceding them, and then we have to take into account in dSUMO that there is potentially a lack of information for vehicles when they are close to the border between partitions. We propose to update the position, speed, etc. of vehicles that just crossed the border to their original node, in order to let the vehicles know that there are obstacles just after the border (see Section 6).

3 Partitioning

The classical first step when setting up a distributed system is the partitioning of processes and data. It consists in splitting instructions and environment and to put it onto the several nodes that will run the distributed system. Some constraints: processing power of the nodes, characteristics of the connections, etc. drive the

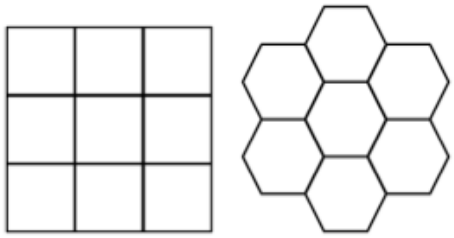
partitioning. Here we cannot partition the processes as every dSUMO node runs a similar SUMO instance. So all we care about is the partitioning of the data and the distribution to the various dSUMO nodes.

Every simulation model in SUMO is composed of a map, a population and detectors. We do not care here about the detectors as setting them and so on is similar to SUMO -- note that we do not provide here a global logging system for the detectors and each computing node will record the outputs of the detectors that it hosts. You can see in Figure 1 a map of the city of Dublin and a possible partitioning of it. Every partition is assigned to a dSUMO node, i.e. its road network and the vehicles that are running on it. Classically, partitioning a map is done using a space partitioning algorithm. But we advocate that our use case (road network) is closer to graph partitioning than classical space partitioning for distributed simulations. The rest of this section addresses the classical partitioning methods, our approach and a solution we presented in a previous publication: *SParTSim* [22].

3.1 Space Partitioning

Distributed simulations usually rely on space partitioning for interest management, i.e., management of communications and synchronisations of agents in a number of nearby space partitions. The agents subscribe to these partitions close to them and receive events and updates that may interest them. This is exactly the scheme used in massively multiplayer online games, where participants (e.g. players' avatars, non-player characters) join and leave partitions as they move on the large virtual environment distributed on several servers [5].

Space partitioning schemes can be divided into two classes: uniform partitioning and non-uniform partitioning. Uniform partitioning split the virtual space into regular, uniform, static partitions, usually rectangles [21] or hexagons [13] (see Figure 3).



(a) Rectangular Zones (b) Hexagonal Zones

Fig. 3. Uniform Partitioning.

Choosing between squares/rectangles and hexagons has garnered a lot of attention and usually depends on application and so on [16, 23]. To overcome the problem of unbalanced load on zones, authors of [17] propose a solution which creates overlays on the simple elements (rectangles or hexagons). This is less regular but allows to avoid to waste some zones when there is no or little activity on them. This approach is

more flexible and can help developers to define precise partition shapes according to terrain, etc., but increases the overhead of managing the areas of interests: finding the neighbouring areas and so on. Figure 6 shows a possible non regular square partitioning over Dublin map.

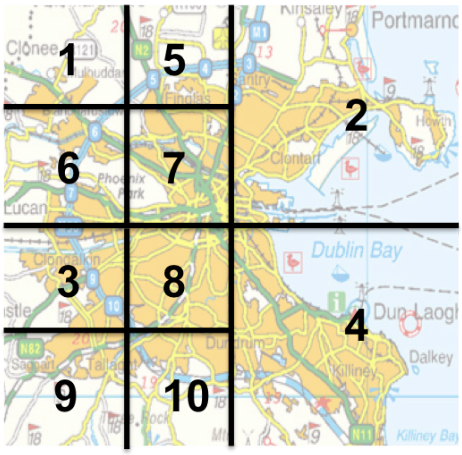


Fig. 4. Non regular partitions.

Figure 5 shows some irregular overlay on a uniform square partitioning.

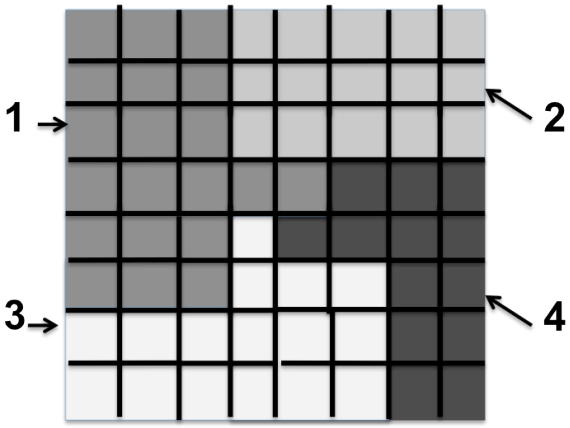


Fig. 5. Irregular overlays.

On the other hand, non-uniform partitioning schemes allow partitions with different shape, size, etc. Most of the time these solutions employ a hierarchical structure to store the relationships between partitions to ease the retrieval and navigation between them. Authors in [2] defines a system where partitions have an arbitrary shape and the relations between them are stored in a binary space

partitioning tree (BSP tree). [18] gives another partitions schemes presentation where they describe four types: quadtree, k-dimensional tree (k-d tree), constrained k-d tree, and region growing. The decision regarding the kind of structure usually depends on some properties that the developers want to achieve: balancing, flexibility, processing-time, etc. There exists some improvements using, for instance, clustering of quad trees [20] to lower the processing time of finding neighbours. Another approach uses Voronoi diagrams [9] which have very nice mathematical properties (all points in the Voronoi region are closer to the centre of its region than the centre of any other region) but are heavy to compute.

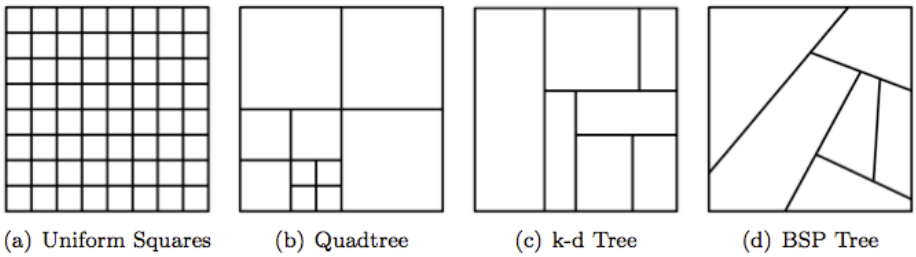


Fig. 6. Non regular partitions.

3.2 Graph Partitioning

Graph partitioning is everywhere in computer science and engineering: distributed computing, computer vision, very large scale integration for circuit layout composition, telephone network design, physical mapping of DNA, route planning, clustering, etc. (e.g., [4, 12]). The general presentation of this problem consists in splitting a graph such that the number of vertices in each partition is balanced and the number of cut edges is minimised. In our case, we obviously interpret that as maximising the load balancing between partitions and minimising the communication, i.e. the number of vehicles that cross the borders between partitions. This is a very complex problem, most instances being NP-hard [7] and many heuristics have been proposed until now [11, 12, 15].

We think that space partitioning techniques described above are not really relevant for urban traffic simulation as the space is composed of a road network. The nature of this road network is such that graph partitioning makes more sense. But they are usually very complex algorithms, may be a little too complex for simple graphs like road networks. Our challenge is then to define a road network partitioning that is as efficient as graph partitioning techniques and as effective as space partitioning ones.

3.3 SParTSim Algorithm

SParTSIM stands for Space Partitioning guided by road network for distributed Traffic Simulations. It defines a hierarchical partitioning based on the road network,

which is by nature hierarchical (there are several levels of roads). In Ireland for instance there are six levels in the road hierarchy: National Primary Roads and Motorways, National Secondary Roads, Regional Roads and three levels of Local Roads. Figure 7 shows three of those levels for Dublin: motorway M50 (strong solid line), National primary roads N1-4, N7 and N11 (tighter solid line), and some local primary roads (dashed lines).

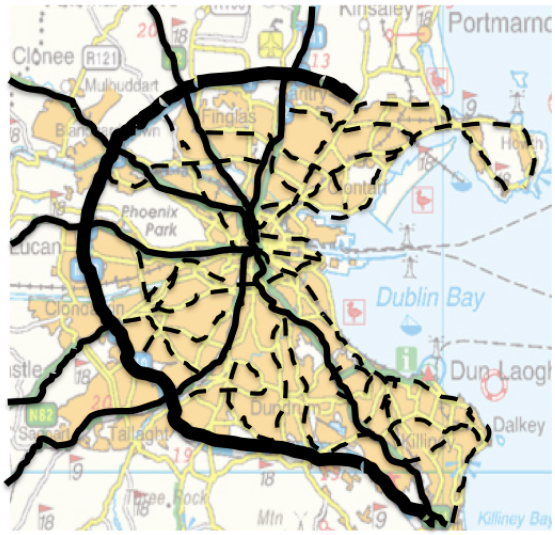


Fig. 7. Three top road levels for Dublin.

SParTSIM uses this hierarchy, following the idea proposed in Metis [11]. In such hierarchical partitioning scheme the complex graph is simplified in a version easily partitioned, and takes gradually back its original size, refining the original partitioning during the process.

SParTSim also applies a region-growing technique: it takes an initial vertex of the road graph and grows a region by adding new nearby vertices. Each region grows in all the directions and competes with others for vertices, until there is no more vertex available. Regions, or partitions, then trade their vertices to balance their size. See our paper [22] for more details.

4 Architecture of dSUMO

In this section, we present the global architecture of our solution, while we show in more details the communication/synchronisation in the next section and the management of the borders, probably the most complex part of distributed simulation, in the second next section. In short, the architecture of dSUMO relies on a few numbers of objects whose purpose is to manage the simulation of a dSUMO node and to communicate with other dSUMO nodes. See Figure 8 for more details.

Every physical machine in dSUMO runs a *dSUMO Container* which is a global environment for the different *dSUMO nodes*, each of them running an instance of SUMO. A classical deployment consists in running as many nodes on a machine as it has cores: e.g., on a dual core machine, there will be a single Container running two nodes. The Container offers the interfaces for communication with other distant nodes (*Server* and *Clients*), as well as the manager for the simulation (*Handler*). Figure 8 presents the different modules and their connections. There is only one Server per Container, but as many Clients as neighbours the dSUMO nodes have (they share the clients). Similarly, only one Handler is needed to command the simulation in the Container. Finally, instances of SUMO are managed by their respective *Runners*. In this section we give some details on each of these modules.

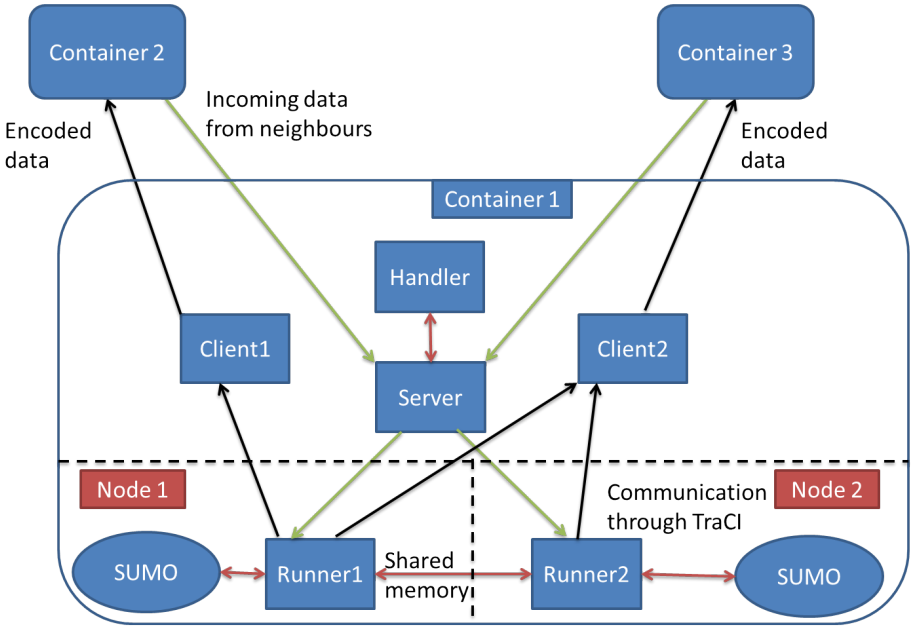


Fig. 8. Architecture of dSUMO.

4.1 Handler

The Handler serves as an interface with the operator (e.g., a user or a script that commands the simulation). It uses the other modules and sends them orders: pause/resume, dump the dSUMO node's state to a file, add/remove cars, stop cars to simulate accidents, etc. It is extensible and aims at providing all the functionalities that you can get from SUMO.

4.2 Runner

The Runner is the most important element of the dSUMO system. This module interacts directly with the SUMO instance through TraCI and does the setup, the

evolution, etc. First, the Runner creates the environment needed to run the distributed simulation: it creates the links with other nodes (distant or local to the Container) and maps the road ends on its partition to these other neighbouring nodes, loads the vehicles in the simulation, etc. Then, when the system is set up, the Runner looks after the vehicles leaving or joining the partition its managing: (i) it instantiates the coming vehicles' route in its partition (ii) it collects information about vehicles close to the border but in other partitions and gives this information to its own vehicles close to the border (iii) it detects when vehicles are leaving its partition and transfers them to other nodes. Finally, it transfers to the each client the data they need to send to their respecting neighbour.

4.3 Client

When a dSUMO container c_i enters the system, i.e. a new machine is used, it also creates one Client per neighbour of its dSUMO nodes $n_1^{c_1}, \dots, n_j^{c_j}$ to handle the connections with them. At the end of every step, the Runners of nodes $n_1^{c_1}, \dots, n_j^{c_j}$ prepare the information they need to send to their neighbours and pass it to the corresponding Clients. When the Clients have all these messages from the Runners, they format the message and send it to the Server they are connected to. The message will be ended by the EoS messages[1].

4.4 Server

Servers are the components responsible for collecting messages from all the Clients of the dSUMO node's neighbours. These messages are translated into TraCI instructions and create new vehicles in the dSUMO node's simulation. Servers also inform TraCI when the simulation can move one step forward, i.e. when they receive the EoS messages[2] and are sure that no event can reach the node before the next step of the simulation.

5 Communication in dSUMO

A car reaching the border between two partitions in the simulation corresponds to a vehicle that needs to be transferred from a node to one of its neighbours. Technically, the dSUMO node's Runner gets the information that a vehicle is reaching the border with another partition and looks up in its dictionary to and the corresponding node. It then sends a message to the relevant Client which forwards the info to the neighbour's Server.

We have implemented the connection between dSUMO Clients and Servers using sockets to ensure the reliability of connection while keeping a good performance.

We have defined several types of message in dSUMO:

---

[1] See later section 5

[2] See later section 5; these messages inform that all the synchronisation events, i.e., vehicles crossing the border between partitions, have been sent.

- A *vehicle message* contains the important information regarding the vehicle crossing the border: *id*, *type* (SUMO definition which contains all the characteristics of the car), *speed*, *route*.
- There are also *bootstrap messages* which synchronise the connection with something like a “ping” to the neighbour's Server.
- When the connection between nodes is established, the Servers acknowledge with a *connection accepted* message.
- *End of Step* (EoS) are messages sent by Clients to their neighbours when all the vehicles have reached the border at the end of a step. They tell the neighbour that nothing can come from the node before the next step.
- The *Back control message* is used to control the replication of a car which just crossed a *border*. It contains the speed that a vehicle will have the next step in order to propagate the impact of a slowing down or a traffic jam.

6 Management of the borders

Managing the border is a critical element in distributed simulations. Vehicles need indeed to know what is after the border in order to make the correct decisions regarding their speed and so on. SUMO for instance has a strong car following model and the knowledge of the position and behaviour of the preceding car is very important for every vehicle. For instance, if there is a traffic jam on node A, close to its border with node B, vehicles coming from B which are closed to cross the border have to know the situation in order to change their route, slow down, stop, etc. If we do not have a very accurate border management in dSUMO then there might be a difference between the classical single CPU SUMO simulation and dSUMO which is not desirable.

6.1 Sending a car

When a car reaches the border of a partition, it needs to be sent to the node managing the nearby partition in order to continue its trip. Runners in dSUMO encode the most relevant information regarding the vehicle such as id, speed, position on the lane, route and type into a message before passing it to a Client who will send it through a socket to the proper neighbour. Once the message reaches the next node's Server, the information is decoded and the vehicle is created inside the SUMO instance by the Runner.

Actually dSUMO does not use the classical SUMO/TraCI method for creating vehicles. The actual implementation of the method that creates vehicles put them in a queue and adds them at the next step, not at the current one, to make sure the vehicles can safely be added. The problem for us is that it would cause a gap in the simulation.

Our solution consists in adding the newly created vehicle to the list of already existing vehicles. This new method allows the vehicles to move at the step $t + 1$ (if t is the step when they cross the border) instead of being added at $t + 1$ and moving at $t + 2$.

6.1 Back-controlling a car

Interest management is probably the most difficult thing with distributed simulations. The problem is that the car following model implemented in SUMO requires every car to know the characteristics (position, speed, etc.) of the car preceding it. Which, translated in dSUMO concepts, means that a car crossing the border between partitions **A** and **B** cannot disappear from **A** if there are cars still following it in **A**.

In dSUMO every road segment split by a border is then duplicated, and exists in both partitions. Let say road segment **RN1** is split by the border between **A** and **B**. **RN1** exists in **A** and **B**, but we say that the lane going from **A** to **B** is managed by **B**, and vice-versa. When car c_1 crosses the border, dSUMO node **A**'s Runner create a message with c_1 , sends it to **A**'s Client responsible for the link with **B**, and this one sends a message to B's Server. The car has crossed the border -- and has disappeared from **A**. Now we do not want cars following c_1 to lose it from sight as its behaviour (speed, etc.) is important. **B** will then sends c_1's characteristics back to **A** at every step, until there is another vehicle that crosses the border or c_1 leaves the segment **RN1**.

Technically, we rely on methods used to move all the cars. We modified them to work for single cars and to have no impact on the overall simulation. They manage to send to the original partition where the car comes from the speed and position, and allows following cars to adapt theirs.

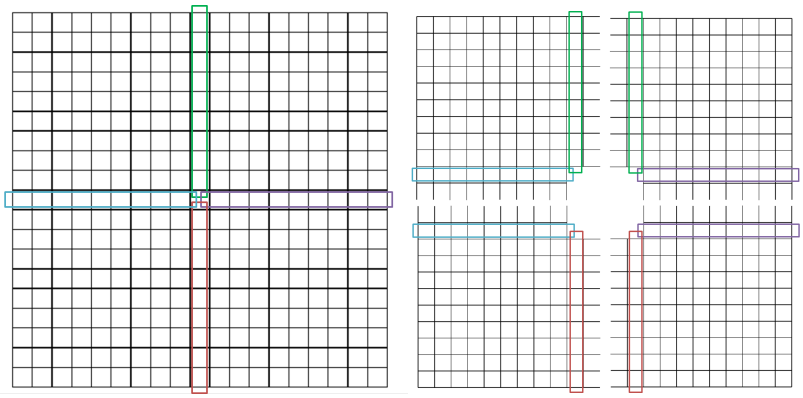
7 Validation

Our validation has two main objectives:

- to evaluate the impact of the distribution on the accuracy of the simulation, i.e., we check if the position of vehicles on dSUMO differs from the one with the centralised SUMO.
- to observe the execution time of dSUMO and to compare it to SUMO.

7.1 Validation Set-up

We use a grid network composed of 20x20 junctions linked by double way-single lane road segments of 200m. The distributed simulations run on partitions of either 10x10 junctions (4 partitions). The road links between junctions are still double way-single lane segments. All vehicles have a North-South or South-North predefined route (starting 40 at a time, on all of the 20 top and bottom junctions each turn). The machines used are either Pentium dual core T4300 (2.1GHz) with 4GB of RAM, or Pentium dual core P6200 (2.13GHz) with 4GB of RAM.



(a) (b)

Fig. 9. (a) Centralised version of the road network (b) Distributed version of the road network.

7.2 Accuracy of dSUMO

To prove that our distributed version of SUMO has no impact on the accuracy of the simulation, we need to show that with the same settings, at every step, the same cars on dSUMO and SUMO are at the same position and have the same speed. We identified two relevant and distinct scenarios which can measure the impact of the distribution:

- A very simple scenario with the basic settings described in the previous section. One car is sent on each vertical road every step. We then stop the simulation at some specific step and compare the position and the speed of every car in SUMO (centralised) with their equivalent in dSUMO.
- We quickly realised that traffic jams just after the borders are the more tricky situations, as cars need to be warned in advance when they reach the border that they will have to stop or slow down. In the second scenario, we stop some cars few steps after they cross the borders, generating traffic congestion. When the jam is propagated to the previous partition, we stop the simulation and compare SUMO and dSUMO.

Note that for this comparative study, we had to set to 0 the random sigma value that gives some variability to vehicles behaviours in SUMO. This sigma value depends on the number of steps the car has spent in the simulation so far (which explains why this value is always the same if you run several times the same experiments with the same settings). Our problem here was that when vehicles pass from one partition to another, they appear as new for the SUMO instance of the dSUMO node and then have a different sigma than their counterparts in the centralised SUMO. Thus generating artificial gaps between SUMO and dSUMO. Null sigma means there is no randomness in acceleration and so on and, centralised and decentralised versions can be compared.

Our accuracy metrics corresponds to the percentage of cars that are at exactly the same position and have the same speed in SUMO (centralised) and dSUMO.

Table 1. Accuracy comparison of SUMO (centralised) and dSUMO.

Scenario	End of simulation (step number)	# of vehicles	Accuracy
Simple	150	2440	100%
Congestion	190	3080	99.38%

Our first attempt to run the first scenario proved to be a failure: it seemed cars are not created at the same absolute position in the distributed SUMO compared to the centralised one. It appeared to be because the road segment does not have the exact same length if they are a cul-de-sac or if they lead to another crossroad. Now, every road segments split during the partitioning becomes a cul-de-sac, and hence the length of the segments was artificially slightly different. We fixed this problem by adding another road segment after the border. As you can see in Table 1 the accuracy is the maximum, which means that the simulations with SUMO or dSUMO are 100% similar.

It is not the case with the congestion scenario though. After investigating the matter, it seems that there is an unexpected safety check in SUMO before a car is created on the new node. It obviously makes sense to create vehicles only if they can be added to the simulation. In our scenario though, SUMO retains some cars, while we know there is room for them: they are present on the previous node at the exact same place, proving that the car has the place to be added on the next node also. We will need more time to see where is this SUMO security/safety test and whether we can remove it safely. Anyway the accuracy is very high and we are rather satisfied.

7.3 Execution Time

As any other distributed system, distributed simulations need a lot of synchronisation and communication. In our case, nodes exchange messages representing cars that are passing the borders, cars that are close to the borders and may have an impact on the simulation, etc. and also messages informing that a step has been processed. And as for any other distributed system, this communication element is an overhead when we compare dSUMO to SUMO (which does not need this synchronisation step). Figure 10 shows the synchronisation time required for simulations with four partitions on the same machine (thus using scheduling between the two cores) and on two different machines. In average, we have around 0.32s, when all dSUMO instances are on the same computer, and around 0.38s, when they are on two remote computers. This is a bigger value than what we expected, and seems to be an issue for our idea of using dSUMO for very large scale or faster than real time simulations. We could argue that in the field of distributed simulations, techniques exist to reduce this synchronization time such as optimistic synchronisation mechanisms [6] (synchronisation is done only every x steps, at the risk of rolling back the simulation). But most of these solutions tend to sacrifice accuracy, which is a hard constraint in our case, and the reason why we pick micro-simulation. Another option could be to use other IPC techniques than sockets (e.g., distributed shared memory) or better communication library than the one we use

currently. Note also that the synchronisation time is constant and seems to be independent of the number of cars we transfer and likely the number of nodes we have in the system.

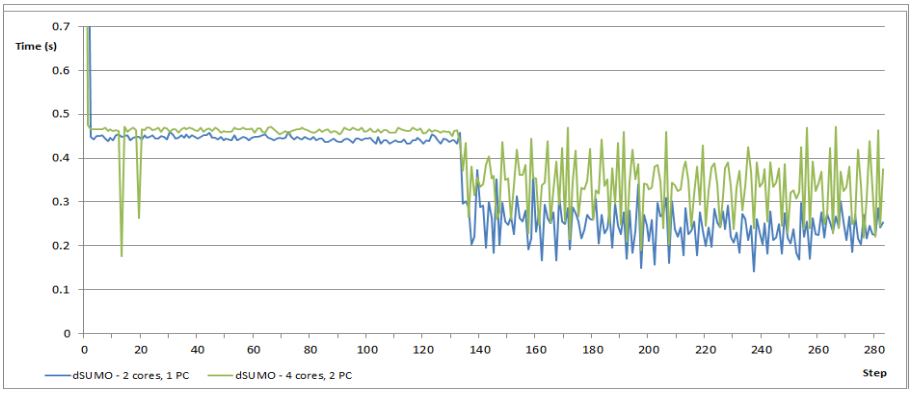


Fig. 10. Evolution of the time required for the synchronisation step.

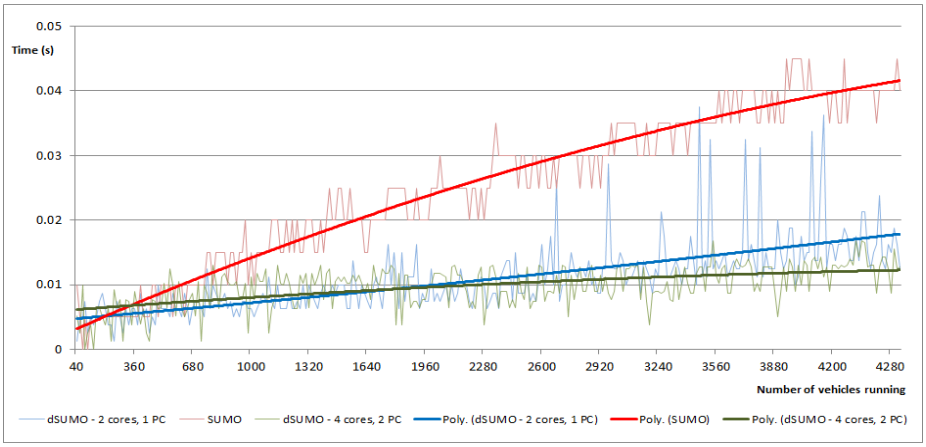


Fig. 11. Evolution of time required per step depending of the number of cars present on the simulation.

Now, if we compare only the execution time of SUMO (reminder: every dSUMO node runs a SUMO instance) we have another perspective on the performance of dSUMO. Figure 11 shows the execution time of SUMO only for (i) SUMO (centralised solution) (ii) dSUMO when 4 nodes run on a single machine (iii) dSUMO when 4 nodes run on two remote machines. We can see that four instances of SUMO running on four different nodes outperform SUMO by an expected factor of about four. Which is very good as it means that for very large simulations SUMO would not scale while adding nodes to dSUMO will minimise the processing time (and remember that synchronisation is a constant). It was of course expected as the time needed by SUMO for each step is strongly dependant on the number of cars and

follow a linear function. With an efficient partitioning algorithm, we can divide the time for each step by the number of nodes we use.

8 Conclusion

In this paper, we have presented dSUMO, a solution towards a distributed version of SUMO. We have listed the possibilities regarding road network partitioning, and defined an architecture where simple elements, e.g. cores of CPUs, can host running instances of SUMO. These nodes are linked to other nodes through sockets and exchange messages corresponding to vehicles transferred, vehicles that are close to the borders and may have an impact on the simulation, end of step, etc. There is no central entity in dSUMO, which makes it scalable and less prone to failures.

Our first experiments show that dSUMO is very accurate, although we have identified an issue that we need to fix before releasing our system: some vehicles are not created by the SUMO instances when they pass from one partition to another, likely for some safety reasons.

Communication is a bigger overhead than we expected and our small evaluations do not show that dSUMO definitely outperforms SUMO. However, if we look at execution time only, independently of synchronization process, then dSUMO reveals its nature: it is distributed and hence the load is balanced over several computing nodes. Which is of course what we expected and makes dSUMO promising anyway. Specially since we observed that synchronization is a constant and does not seem to vary greatly with the number of cars exchanged or the number of nodes.

We expect to be able to release soon a first version of dSUMO, for users feedback and so on.

References

1. Dublin city council workplace travel plan. http://www.dublincity.ie/RoadsandTraffic/Documents/DCC_Travel_Plan_v2_1[1].doc
2. Barrus, J.W., Waters, R.C., Anderson, D.B.: Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments. In: VRAIS, pp. 204–213 (1996)
3. Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D.: Sumo - simulation of urban mobility: An overview. In: SIMUL 2011, The Third International Conference on Advances in System Simulation. Barcelona, Spain (2011)
4. Bhatt, S.N., Leighton, F.T.: A framework for solving VLSI graph layout problems. Journal of Computer and System Sciences 28(2), 300–343 (1984)
5. DMSO: High Level Architecture Interface Specification Version 1.3 (1998). URL http://hla.dmso.mil
6. Fujimoto, R.M.: Parallel and Distribution Simulation Systems, 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1999)

7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
8. Habitat, U.: State of the world's cities 2010/2011 - cities for all: Bridging the urban divide. http://www.unhabitat.org/pmss/listItemDetails.aspx?publicationID=2917
9. Hu, S.Y., Chen, J.F., Chen, T.H.: VON: A Scalable Peer-to-Peer Network for Virtual Environments. IEEE Network 20, 22–31 (2006)
10. Joerer, S., Dressler, F., Sommer, C.: Comparing apples and oranges?: trends in ivc sim- ulations. In: Proceedings of the ninth ACM international workshop on Vehicular inter- networking, systems, and applications, VANET '12, pp. 27–32. ACM, New York, NY, USA (2012).
11. Karypis,G.,Kumar,V.:Afastandhighqualitymultilevelschemeforpartitioningirregular graphs. SIAM J. Sci. Comput. 20(1), 359–392 (1998)
12. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal 49(2), 291–307 (1970)
13. Macedonia, M.R., Zyda, M.J., Pratt, D.R., Brutzman, D.P., Barham, P.T.: Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. In: VRAIS, pp. 2–10 (1995)
14. Morgenroth, E.L.W.: Analysis of the economic, employment and social profile of the greater dublin region. http://www.dublinpact.ie/pdfs/profile.pdf (2001)
15. Pothen, A., Simon, H.D., Liou, K.P.: Partitioning sparse matrices with eigenvectors of graphs. SIAM Journal on Matrix Analysis and Applications 11(3), 430–452 (1990)
16. Prasetya, K., Wu, Z.D.: Performance Analysis of Game world Partitioning Methods for Multiplayer Mobile Gaming. In: ACM SIGCOMM Workshop on Network and System Support for Games, pp. 72–77 (2008)
17. Srinivasan, S., Supinski, B.R.D.: Multicasting in DIS: A Unified Solution. In: Electronic Conference on Scalability in Training Simulation (1995)
18. Steed, A., Abou-Haidar, R.: Partitioning Crowded Virtual Environments. In: VRST, pp. 7–14 (2003)
19. Tanenbaum, A.S., van Steen, M.: Distributed systems - principles and paradigms (2. ed.). Pearson Education (2007)
20. Van Hook, D.J., Rak, S.J., Calvin, J.: Approaches to RTI Implementation of HLA Data Distribution Management Services. In: Workshop on Standards for the Interoperability of Distributed Simulations (1997)
21. Van Hook, D.J., Rak, S.J., Calvin, J.O.: Approaches to Relevance Filtering. In: Workshop on Standards for the Interoperability of Distributed Simulations, pp. 26–30 (1994)
22. Ventresque, A., Bragard, Q., Liu, E., Nowak, D., Murphy, L., Theodoropoulos, G., Liu, Q.: Spartsim: A space partitioning guided by road network for distributed traffic simulations. In: DS-RT. Dublin, Ireland (2012)
23. Yu, A.P., Vuong, S.T.: MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In: workshop on Network and oper- ating systems support for digital audio and video, pp. 99–104 (2005)